Python

Lists and Dictionaries

@kabirbaidhya

Reflections

What we know already

After the previous sessions we know:

1. Python basics - Variables, types, operations
2. Strings and Formatting

Lists

Lists

List is one of the most common and versatile data structures we use in python when it comes to storing a group of items or values.

You can construct a list in python simply as a list of comma-separated values (items) in between square brackets.

Like this:

```
numbers = [1, 2, 3, 4, 5, 6, 7]
```

Can Contain Anything

Elements in a list can be of different data types and can contain another (nested) lists as well.

```
# Can contain group of random dissimilar elements.
misc_list = [1, 'foo', '2', 77.00, [5, 6, 7], True]

# Can contain nested lists.
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7. 8, 9]
]
```

Common Operations

Like strings, lists are sequences too and thus support the operations that sequences in python support.

Operation	Result
x in s	True if an item of s is equal to x, else False
x not in s	False if an item of s is equal to x, else True
s + t	the concatenation of s and t
s * n or n * s	equivalent to adding s to itself n times
s[i]	ith item of s, origin 0
s[i:j]	slice of s from i to j
s[i:j:k]	slice of s from i to j with step k

Common Operations

Operation	Result
len(s)	length of s
min(s)	smallest item of s
max(s)	largest item of s
s.index(x[, i[, j]])	index of the first occurrence of x in s (at or after index i and before index j)
s.count(x)	total number of occurrences of x in s

For Instance

```
>>> numbers = [1, 2, 3, 4, 5, 6, 7]
>>> len(numbers)
7
>>> min(numbers)
1
>>> max(numbers)
7
>>> numbers.count(5)
1
>>> 7 in numbers
True
>>> 8 in numbers
False
>>> 0 not in numbers
True
```

Concatenation

Lists do support even concatenation just the way strings do with the `+` operator.

```
>>> numbers + [8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Example 1

It's possible to mutate or change the lists like this:

```
names = ['John Doe', 'Jane Doe', 'Johnny Turk']
# Change the first name in the list
names[0] = 'Foo Bar'
print('Names now:', names)

# Append some more names
names.append('Molly Mormon')
names.append('Joe Bloggs')
print('Names finally:', names)
print('Last name in the list: %s' % names[-1])

# You can join lists using str.join() method
joined_names = '\n'.join(names)
print('\nList of names:')
print(joined_names)
```

List Methods

The list object support the following methods.

Method	Description
append(x)	Add an item to the end of the list.
insert(i, x)	Insert an item at a given position.
remove(x)	Remove an item from the list whose value equals to `x`. An error is thrown if the value is not found in the list.
pop([i])	Remove the item at `i`th position in the list, and return it. If index `i` is not, the last item is removed and returned

List Methods

Method	Description
clear()	Remove all items from the list.
index(x[, start[, end]])	Return the first index whose value is `x`. Throws an error if the value is not found in the list.
count(x)	Count the number of times the value `x` appears in the list.
sort(x)	Sort the items of the list.
reverse()	Reverse the items of the list.
copy()	Return a shallow copy of the list.

Check the official docs for lists more more information.

Try them out

Let's try these methods as well.

```
>>> names.insert(0, 'Mark Joe')
>>> names
['Mark Joe', 'Foo Bar', 'John Doe', 'Jane Doe', 'Johnny Tur
>>> names.remove('Foo Bar')
>>> names
['Mark Joe', 'John Doe', 'Jane Doe', 'Johnny Turk', 'Molly
>>> names.pop()
'Joe Bloggs'
>>> names.pop(0)
'Mark Joe'
>>> names
['John Doe', 'Jane Doe', 'Johnny Turk', 'Molly Mormon']
>>> names.sort()
```

List Comprehensions

List comprehension is a pythonic way of creating lists in a concise manner based upon the results of some operations or certain conditions.

List comprehensions is one of the most popular features of python lists.

```
# Create a list of squares of numbers upto 10
squares = [x**2 for x in range(10)]

print('Squares:', squares)
```

Example 2

```
# You can create lists using existing lists.
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_numbers = [x for x in numbers if x % 2 ==0]
odd_numbers = [x for x in numbers if x % 2 !=0]

print('Numbers:', numbers)
print('Even numbers:', even_numbers)
print('Odd numbers:', odd_numbers)
```

Example 3

```
# You can even create new lists by processing existing list
words = ['this', 'is', 'just', 'a', 'test']
capitalized_words = [x.capitalize() for x in words]

print('Words:', words)
print('Capitalized Words:', capitalized_words)

# Can use it for filtering the list items as well.
words = ['hello', 'world', 'foo', 'bar', 'test', 'python',
short_words = [x for x in words if len(x) <= 3]
other_words = [x for x in words if x not in short_words]
words_with_e = [x for x in words if x.count('e') >= 1]

print('Words:', words)
print('Short Words:', short_words)
print('Other Words:', other_words)
print('Words with "e":', words_with_e)
```

Looping

Looping through list is really simple in python. You can use `for` loop for looping through lists.

```
names = ['John Doe', 'Jane Doe', 'Johnny Turk']

print('Names:')

for name in names:
    print(' - %s' % name)
```

Example 5

You might often want to check if the list is empty.
Usually the list is dynamically generated.
You can do that by checking no. of items in the list is zero or not.

```
my_list = []

if len(my_list) == 0:
    print('No items on the list.')
else:
    print(my_list)
```

Dictionaries

Dictionaries

Another most common data structure used in python is a dictionary.
The main difference between sequence types like strings & lists and dictionaries is that sequences are indexed by range of numeric indexes but dictionaries are indexed by keys.

Any immutable data type can be used as keys in dictionaries, usually they are strings and numbers.

You can always think of a dictionary as a set of key value pairs.

Example 6

You can create a dictionary like this:

```
user_info = {
    'name': 'Kabir Baidhya',
    'email': 'kabirbaidhya@gmail.com',
    'address': 'Kathmandu, Nepal'
}

# Accessing key from a dict is just similar to lists.
print('Name: %s' % user_info['name'])
print('Email: %s' % user_info['email'])
print('Address: %s' % user_info['address'])
```

Example 7

Since dictionaries are mutable types you can mutate them just like lists.

```
user_info['name'] = 'Kabir'
user_info['email'] = user_info['email'].replace('@gmail.com

# If the key doesn't already exists it would create a new k
user_info['dob'] = '1992-07-30'

# And you can store any type of values inside a dict. Even
user_info['hobbies'] = ['Music', 'Travelling', 'Coding']

print(user_info)
```

Example 8

You can also create a list of dictionaries.

```
data = [
    {
        'name': 'Kabir Baidhya',
        'email': 'kabirbaidhya@gmail.com'
    },
    {
        'name': 'John Doe',
        'email': 'johndoe@example.com'
    }
]

# Print information from the dictionary
print('Name: %s' % data[1]['name'])
print('Email: %s' % data[1]['email'])
```

Common Operations

Operation	Description
len(d)	Return the number of items in the dictionary `d`.
d[key]	Access/Return the item of dictionary identified by key `key`. An error is thrown if `key` is not found in the dictionary.
d[key] = value	Set a value in the dictionary identified by `key`.
del d[key]	Remove the item with key `key` from the dictionary. An error is thrown if `key` does not exists
copy()	Return a shallow copy of the dictionary.
clear()	Remove all the items from the dictionary.

Common Operations

Operation	Description
key in d	Check if the `key` exists in the dictionary. Return `True` or `False`.
key not in d	Check if the `key` doesn't exist in the dictionary i.e just the opposite of `key in d`. Return `True` or `False`.

Read more about dictionaries here.

Exercises

Exercise 1

Store a list of at least 20 words in a list. Ask the user to enter a string(partial) and print out the list of suggestions based on whether or not the word starts with the string entered.

Note: the suggestion should be case-insensitive. (Hint: List comprehension).

Exercise 2

Store a list of user information in a list of dictionaries. Each user's information would contain: first & last name, email and address. Ask the user to input an email address. Print the first user's information found by that email address. Print "Email not found" message if user with email not found. (Hint: List comprehension)

Read More?

Links

1. https://docs.python.org/3/tutorial/datastructures.html#dictionaries
2. https://docs.python.org/3/library/stdtypes.html#typesmapping
3. https://docs.python.org/3/tutorial/introduction.html#lists
4. https://docs.python.org/3/tutorial/datastructures.html#more-on-lists

Thank You

@kabirbaidhya

kabirbaidhya@gmail.com